

OptCon: An Adaptable SLA-Aware Consistency Tuning Framework for Quorum-based Stores

Subhajit Sidhanta*, Wojciech Golab[†], Supratik Mukhopadhyay* and Saikat Basu*

*Louisiana State University, Baton Rouge, Louisiana, USA, Email: {ssidha1, supratik, saikat}@csc.lsu.edu

[†]University of Waterloo, Waterloo, Ontario, Canada, Email: wgolab@uwaterloo.ca

Abstract—Users of distributed datastores that employ quorum-based replication are burdened with the choice of a suitable client-centric consistency setting for each storage operation. The above matching choice is difficult to reason about as it requires deliberating about the tradeoff between the latency and staleness, i.e., how stale (old) the result is. The latency and staleness for a given operation depend on the client-centric consistency setting applied, as well as dynamic parameters such as the current workload and network condition. We present OptCon, a novel machine learning-based predictive framework, that can automate the choice of client-centric consistency setting under user-specified latency and staleness thresholds given in the service level agreement (SLA). Under a given SLA, OptCon predicts a client-centric consistency setting that is *matching*, i.e., it is weak enough to satisfy the latency threshold, while being strong enough to satisfy the staleness threshold. While manually tuned consistency settings remain fixed unless explicitly reconfigured, OptCon tunes consistency settings on a per-operation basis with respect to changing workload and network state. Using decision tree learning, OptCon yields 0.14 cross validation error in predicting matching consistency settings under latency and staleness thresholds given in the SLA. We demonstrate experimentally that OptCon is at least as effective as any manually chosen consistency settings in adapting to the SLA thresholds for different use cases. We also demonstrate that OptCon adapts to variations in workload, whereas a given manually chosen fixed consistency setting satisfies the SLA only for a characteristic workload.

I. INTRODUCTION

Many quorum-based distributed data stores [22, 28, 8], allow the developers to explicitly declare the desired *client-centric consistency* setting (i.e., consistency observed from the viewpoint of the client application) for an operation. Such systems accept the client-centric consistency settings for an operation in the form of a runtime argument, typically referred to as the *consistency level*. The performance of the system, with respect to a given operation, is affected by the choice of the consistency level applied [35].

From the viewpoint of the user, the most important performance parameters affected by the consistency level applied are the latency and client-observed *consistency anomalies*, i.e., anomalies in the result of an operation observed from the client application, such as stale reads. Consistency anomalies are measured in terms of the client-centric *staleness* [36], i.e., how stale (old) is the version of the data item (observed from the client application) with respect to the most recent version. According to the consistency level applied, the system waits for coordination among a specific number of replicas containing copies (i.e., versions) of the data item accessed by the given operation [22]. If the system waits for coordination among a smaller number of replicas, the chance of getting a

stale result (i.e., an older version) increases. Also, the latency for the given operation depends on the waiting time for the above coordination; hence, in turn, depends on the consistency level applied. For example, a weak consistency level for a read operation in Cassandra [22], like READ ONE, requires only one of the replicas to coordinate successfully, resulting in low latency and high chances of a stale read. Hence, while choosing the consistency level, developers must consider how this choice affects the latency and staleness for a given operation.

The chosen consistency level must be *matching* with respect to the latency and staleness thresholds specified in the given service level agreement (SLA), i.e., it must be weak enough to satisfy the latency threshold, while being strong enough to satisfy the staleness threshold. Consider a typical use case at Netflix [10] where a user browses recommended movie titles. Such use cases require real-time response [19]. Hence the SLA typically comprises low latency and higher staleness thresholds. For the given SLA, workload, and network state, the matching choice is a weak read-write consistency level (like ONE/ANY in Cassandra). If the developer applies stronger consistency level, the resulting high latency might violate the SLA.

With the current state-of-the-art [35], the developers have to manually determine a matching consistency level for a given operation at development time. Reasoning about the above choice is difficult because of: 1) the large number of possible SLAs, 2) unpredictable factors like changing network state and varying workload that impact the latency and staleness, and 3) the absence of a well-formed mathematical relation connecting the above parameters [2]. This makes automated consistency tuning under latency and staleness thresholds in the SLA a highly desirable feature for quorum-based datastores.

We present OptCon¹, a novel framework that automatically determines a matching consistency level for a given operation under a given SLA. Due to the absence of a closed-form mathematical model capturing the impact of the consistency level, current workload, and network state, on the observed latency and staleness, OptCon applies machine learning [15] to train a model for *predicting* a matching consistency level under the given SLA, workload, and network state. For the Netflix use case, taking into account the read-heavy workload, the current network state, and the SLA thresholds, OptCon predicts a weak consistency level. The contributions of this paper are:

¹The project is partially supported by Army Research Office (ARO) under Grant W911NF1010495. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the ARO or the United States Government.

- We introduce OptCon, a novel machine learning-based framework, that can automatically predict a matching consistency level that satisfies the latency and staleness thresholds specified in a given SLA, i.e., the predicted consistency level is weak enough to satisfy the latency threshold, and strong enough to satisfy the staleness threshold. Using decision tree learning, OptCon yields a cross validation error of 0.14 in predicting a matching consistency level under the given SLA.
- Experimental results demonstrate that OptCon is at least as effective as any manually chosen consistency level in adapting to the different latency and staleness thresholds in SLAs. Furthermore, we also demonstrate experimentally that OptCon surpasses any manually chosen fixed consistency level in adapting to a varying workload, i.e., OptCon satisfies the SLA thresholds for variations in the workload, whereas a manually chosen consistency level satisfies the SLA for only a subset of workload types.

II. MOTIVATION

A. Choice of the SLA Parameters

Following prior research by Terry et al. [35], we include latency and staleness in the SLA for operations on quorum-based stores. The choice of consistency level directly affects the latency for a given operation on a quorum-based datastore [22]. While Terry et al. [35] use categorical attributes for specifying the desired consistency (such as read-my-write, eventual, etc.) in the SLA, we use a more fine-grained SLA, that accepts threshold values for the client-centric staleness [16]. Golab et al. [16] demonstrate that both the proportion and severity of stale results increases from stronger to weaker consistency levels. The use of a client-centric staleness metric in the SLA enables the developer to specify, on a per-operation basis, the exact degree of staleness of results, that a given client application can tolerate.

Following Terry et al. [35], we do not include throughput in the SLA. But it is still desirable to have throughput [9] as a secondary optimization criterion, once the SLA thresholds are satisfied. Depending on the SLA, a set of consistency levels can be matching with respect to the latency and staleness thresholds given in the SLA. Consider a real-time application (such as an online shopping cart) that demands moderately low latency and tolerates relatively higher staleness in the SLA. In such cases, any weaker consistency level (like ONE or ANY in Cassandra) can yield latency and staleness values within the given SLA thresholds. In such cases of multiple matching consistency levels, OptCon chooses the one that maximizes throughput. Also, we do not consider parameters like keyspace size, replication factor, and certain other server-centric parameters in our model, since our focus is optimizing client-centric performance [16].

B. Motivation for Automated Consistency Tuning

The large number of possible use cases [9], each comprising different SLA thresholds for latency and staleness, makes manual determination of a matching consistency level

a complex process. The latency and staleness are also affected [2] by the following independent variables (parameters): 1) the packet count parameter, i.e., the number of packets transmitted during a given operation, represents the network state [22], and 2) the *read proportion* (i.e., proportion of reads in the workload) and thread count, that represent the workload characteristics. Manually reasoning about all of these parameters combined together is difficult, even for a skilled and experienced developer. Further, unpredictable variations in workload, network congestion, and node failures may render a particular consistency level, manually pre-configured at development time, infeasible at runtime [26, 35].

III. CHALLENGES IN AUTOMATING CONSISTENCY TUNING

Currently, there is no closed-form mathematical model [2] that represents the effect of the consistency level on the observed staleness and latency, with respect to the independent variables. Bailis et al. [2] base their work upon the simplifying assumption that writes do not execute concurrently with other operations. Hence, it is not clear from [2] how to compute the latency and staleness from a real workload. Pileus and Tuba [35, 1] are the only systems that provide fine-grained consistency tuning using SLAs. But, instead of predicting, these systems perform actual trials on the system, and select the consistency level corresponding to the SLA row that produces minimum resultant utility, based on the trial outcomes. The trial-based technique can produce unreliable results due to the unpredictable parameters like network conditions and workload that affect the observed latency and staleness. Thus, predictions based on the outcomes of the trial phase may be unsuitable in the actual running time of the operation. Sivaramakrishnan et al. [33] use a static analysis approach to determine the weakest consistency level that satisfies the correctness conditions declared in the contract. They do not consider the tradeoff between staleness and latency, and cannot dynamically adapt to varying workload and network state.

IV. DESIGN OF OPTCON

A. Design Overview

In the absence of a closed-form mathematical model relating the consistency level to the observed latency and staleness, OptCon leverages machine learning-based prediction techniques [15], following the footsteps of prior research [21, 38, 32]. OptCon is the first work that trains on historic data, and learns a model \mathcal{M} relating the consistency level and the input parameters (i.e., the workload and network state) to the observed latency and staleness. The model \mathcal{M} can predict a matching consistency level with respect to the given SLA thresholds for latency and staleness, under the current workload and network state. A matching consistency level is weak enough to satisfy the latency threshold, and simultaneously strong enough to satisfy the staleness threshold. Our definition of matching consistency level is a modified version of the *correct* consistency level in QUELA [33]. For multiple consistency levels satisfying the given SLA thresholds, OptCon predicts the consistency level that maximizes the throughput.

In contrast to program verification-based static (compile time) approaches [33], OptCon provides on the fly predictions

based on dynamic (runtime) values of the input parameters, i.e., the current workload and the network state (Section II-B). Thus, OptCon tunes the datastore according to any dynamic variations in the workload and any given SLA. Furthermore, unlike the state-of-the-art trial-based techniques that base decisions on the values of the parameters obtained during the trial phase [35, 1], OptCon provides reliable predictions (see Section V-G and V-H), taking into consideration the actual runtime values of the input parameters.

Average Latency	Staleness
$\leq 100\text{ms}$	$\leq 5\text{ms}$
$\leq 50\text{ms}$	$\leq 10\text{ms}$
$\leq 25\text{ms}$	$\leq 15\text{ms}$

TABLE I: Example subSLAs

Like Terry et al. [35], users provide latency and staleness thresholds to OptCon in form of an SLA, illustrated in Table I. Each SLA consists of rows of *subSLAs*, where each subSLA row, in turn, comprises two columns containing the thresholds for latency and staleness, respectively. Instead of categorical attributes [35], OptCon uses threshold values for staleness [16] in the subSLAs. Unlike Terry et al., subSLAs in OptCon do not have a utility column. Also, subSLAs are not necessary ordered. If the thresholds for a particular subSLA are violated, OptCon marks the operation as failed with respect to the given subSLA. The handling of such failure cases will depend on the application logic, and is not part of the scope of this paper.

Instead of modifying the source code of distributed datastores, OptCon is designed as a wrapper over existing distributed storage systems. We have implemented and tested OptCon on Cassandra, which follows the Dynamo [13] model. Among the Dynamo-style systems, while Cassandra organises the data in the form of column families, Voldemort [34] and Riak [28] are strictly key-value stores. Also, Cassandra and Riak are designed to enable faster writes, whereas Voldemort enables faster reads. But the principles governing the internal synchronization mechanisms are similar for all these systems. Hence the consistency level choices affect the observed latency and staleness in a similar fashion for all these systems. Thus, OptCon can potentially be integrated with any Dynamo-style system.

B. Architecture



Fig. 1: The architecture of OptCon: rectangles denote modules and the folded rectangle denotes dataset.

OptCon (refer to Figure 1) consists of the following modules: 1) The Logger module records the independent variables (Section II-B), the observed latency, and staleness, obtained using experiments performed on the datastore with benchmark workloads. It collates these parameters into a training data

corpus, which acts as the training dataset. 2) The Learner module learns the model \mathcal{M} (refer Section V-C) from the training dataset, and predicts a matching consistency level that maximizes the throughput. 3) The Client module calls the other modules, and executes the given operation on the datastore.

During the training phase, the Client module runs a simulation workload on the given quorum-based datastore. The Client calls the Logger module (Figure 1) to collect the independent variables (parameters) and the observed parameters for the operation from the JMX interface [22], and appends these parameters into the training data. The Client then calls the Learner module, which trains the model \mathcal{M} from the training data applying machine learning techniques. During the prediction phase, the Client calls the Learner, with runtime arguments comprising the subSLA thresholds and the current values of the independent variables. The subSLA (and also the SLA) can be varied on a per-operation basis. The Learner predicts a matching consistency level from the learnt model \mathcal{M} . The Client performs the given operation on the datastore with the predicted consistency level.

C. Design of the Logger Module

1) *Model Parameters Collected by the Logger:* For the reasons explained in Section II-A, the latency L , i.e., time delay for the operation, and client-centric staleness S , are considered as the model parameters that need to satisfy a given SLA. The throughput T is considered as a secondary optimization criterion. As discussed in Section II-A, server centric parameters, like replication factor and keyspace size, are not considered. Following the reasoning in Section II-B, the independent variables (parameters) for learning the model \mathcal{M} are: 1) the read proportion (RW) in the operation, 2) the thread count (Tc), i.e., the number of user threads spawned by the client, 3) the packet count (P), i.e., the number of network packets transmitted during the operation, and 4) the consistency level C , provided as a categorical attribute containing attribute values specific to the given datastore.

Most operations on quorum based stores are not instantaneous, but are executed over certain time intervals [16]. Following [2], the Logger uses the average latency over a time interval of one minute for measuring L . The Logger computes S in terms of the Γ metric of Golab et al. [16]. As demonstrated in the above work, Γ is preferred over other client-centric staleness measures for its proven sensitivity to workload parameters and consistency level. Section V-E establishes the significance of the above parameters with respect to the OptCon model. Section V-F demonstrates the accuracy of the model comprising the that the above parameters.

2) *Computation of Γ as the Metric for Client-centric Staleness:* The metric Γ is based upon Lamport’s *atomicity* property [24], which states that operations appear to take effect in some total order that reflects their “happens before” relation in the following sense: if operation A finishes before operation B starts then the effect of A must be visible before the effect of B. We say that a trace of operations recorded by the logger is Γ -atomic if it is atomic in Lamport’s sense, or becomes atomic after each operation in the trace is transformed by decreasing its starting time by $\Gamma/2$ time units, and increasing its finish time by $\Gamma/2$ time units. In this context the start and finish times are

shifted in a mathematical sense for the purpose of analyzing a trace, and do not imply injection of artificial delays; the behavior of the storage system is unaffected.

The *per-key* Γ score quantifies the degree of client-centric staleness incurred by operations applied to a particular key. We considered average per-key Γ , but settled for percentile per-key Γ since it takes into account the skewed nature of the workloads.

D. Design of the Learner Module

In OptCon, building the model \mathcal{M} from training examples (Figure 1) is a one-time process, i.e., the same model can be reused for predicting consistency levels for all operations. Even with node failures, the model may not need to be regenerated since the training data remain unchanged, only the failed predictions, resulting due to node failure, need to be rerun. However, the model will need to be retrained in certain cases where the behavior of the system with respect to the latency and staleness changes, such as when a storage system receives a software upgrade.

1) *Overview of Possible Learning Techniques:* We provide a brief overview of the possible Learning techniques and their applicability to OptCon. We describe the implementation details, including the various configuration parameters, of each learning algorithm in Section V-D. Performance analysis and insights gathered from each technique are given in Section V-F2. We consider Logistic regression and Bayesian learning as they can help visualize the significance of the model parameters and the dependency relations among these parameters, and thus can provide intuition to the developer for developing complex and more accurate models [15]. We also consider Decision Tree, Random Forest, and Artificial Neural Networks (ANN), since they can produce more accurate predictions, being directly computed from the data. We consider these approaches in the order of their performance given in Table III, in terms of various model selection metrics. We leave the choice of a suitable learning technique to the developer, rendering flexibility to the framework. Based on our evaluation of the learning algorithms, developers can choose the learning technique that best suits the respective application domain and use case.

With logistic regression approach, we fit two logit (i.e., logistic) functions for the two dependent variables L and S as follows: $\text{logit}(\pi(L)) = \beta_0 + \beta_1 C + \beta_2 RW + \beta_3 P + \beta_4 Tc$ and $\text{logit}(\pi(S)) = \beta_4 + \beta_5 C + \beta_6 RW + \beta_7 P + \beta_8 Tc$, where C is the applied consistency level, L is the observed latency, S is the observed staleness, RW is the proportion of reads, P is number of packets transmitted, and Tc is the number of threads during an operation. β_i are the coefficients estimated by iterative least square estimation, and π is the likelihood function. Using ordinary least square estimation, we iteratively minimize a loss function given by the difference of the observed and estimated values, with respect to the training dataset. For eliminating overfitting, we perform L^1 regularization with the Lasso method. Next we consider decision tree learning algorithm because of its simplicity, speed, and accuracy. The given problem can be viewed as a classification problem, which classifies the training dataset into classes based on whether the observed latency and staleness

(corresponding to each row) fall within the given SLA thresholds. In fact, the problem statement, i.e., choosing a matching consistency level, can be viewed as a classification problem. The modelling technique using Decision Tree comprises the following phases: 1) **Labelling:** On the fly computations for maximizing T (i.e., the secondary optimization criteria) in the prediction phase would introduce considerable overhead. Since the latency is considered before T is maximized, the additional overhead may result in violation of the latency threshold in the SLA. Hence, OptCon performs the computations for T in the labelling phase itself. Using exhaustive search, we label each row in the training dataset with the highest T corresponding to a given values of RW , Tc and P , such that the SLA parameters L and S are within the given SLA thresholds. 2) **Training:** Next we apply decision tree learning for training a model \mathcal{M} from the labelled dataset. Use of error pruning techniques mitigates issues of overfitting [15]. Next, we consider the random forest algorithm that applies ensemble learning to bootstrap multiple weak decision tree learners to generate a strong classifier. It iteratively chooses random samples with replacement from the training dataset, and trains a decision tree on each of the samples. We take the average of the predictions from each decision tree to obtain the final prediction with respect to a test dataset. For the Bayesian approach, we make the following simplifying assumptions to make our problem suitable for Bayesian learning: A1) a prior logistic distribution exists for each of RW , Tc , P , and C (see Section IV), A2) the posterior distributions for L , and S are conditionally dependent on the joint distribution of $\langle RW, Tc, P, C \rangle$, and A3) the target features L and S are conditionally independent. With these approximations, we plug-in the resultant dependency graph from the Logistic Regression approach as input to the Bayesian learner. Despite these assumptions and approximations, the Bayesian technique can provide intuition to the developers, that can be used to build a more accurate model. Next, we consider Artificial Neural Networks (ANN) for learning a model \mathcal{M} . Though the training phase for ANN is not a factor (since training occurs once), the overhead for the prediction phase is still considerably high (Section V-C) due to model complexity.

V. IMPLEMENTATION AND EVALUATION

A. Experimental Setup

Following [16], we have run our experiments on a testbed of 20 Amazon Ec2 small instances, located in the same Ec2 region, running Ubuntu 13.10, loaded with Cassandra with a replication factor of 5. Our models are learnt from a training dataset comprising 23040 rows, and a testing dataset of 2560 rows, generated by running varying YCSB [11] workloads. We have used the NTP (Network Time Protocol) protocol implementation from ntp.org to bring down the maximum clock skew to 5 ms.

B. Latency and Staleness Ranges Used in the subSLAs

Studies [14] suggest that the latency for standard web applications falls in the range of 75-140 ms. The Pileus system [35], which includes latency in subSLAs as well, uses values between 200 to 1000 ms as the latency threshold. Further reducing the lower bound to favor availability, we choose any value between 101 and 150ms as a candidate for the threshold

of latency in OptCon. Following the observations from the Riak deployment at Yammer [17], Bailis et al. [2] uses a range of 1.85-13.6 ms for the staleness bounds, given as the t-visibility values. Hence, following [2], we choose a value within the above range as the staleness threshold. Rounding the bounding values for the ranges given above, our subSLAs use values in the ranges 101-150 ms and 1-13 ms as thresholds for L and S , respectively.

C. OptCon Framework Overhead

Training the model being a one-time process, the latency overhead due to the training phase is negligible during an operation. However, there is still considerable overhead due to the prediction phase, especially with a large training dataset. The prediction phase spans from the call to the Learner module till the predicted consistency level is returned by the Learner module. The average, variance and standard deviation of the overhead are 1 ms, 0.25, and 0.37, respectively, with decision tree learning. The average overhead with different learning techniques is given in Table III.

D. Implementation of the Modules: Logger and Learner

OptCon is developed as a Java-based wrapper over Cassandra v2.1.0. The source code can be found in the github repositories: <https://github.com/ssidhanta/YCSBpatchpredictconsistency/>, <https://github.com/ssidhanta/TrainingModelGenerator/>, and <https://github.com/ssidhanta/HectorClient/>. The Logger module is implemented as an extension over the YCSB 0.1.4 [11] framework. It runs as a daemon on top of the datastore, listening for the parameters using the built-in JMX Interface. Logistic Regression, SVM, and Neural Network implementations of the Learning module use Matlab 2013b's statistical toolbox. Decision Tree and Random Forest are implemented using Java APIs from Weka, an open source machine learning suite. The Bayesian approach uses Infer.net, an open source machine learning toolbox under the .NET framework. We summarize the important configuration parameters in the Learner implementations. Logistic regression uses a minimization function with additional regularization weights, multiplied by a tunable L^1 norm term Λ . We set $\alpha = 0.05$ and $\Lambda = 25$. Decision tree uses a confidence threshold of 0.25 for pruning, and uses random data shuffling with seed = 1. Random forest bootstraps 100 decision trees, and introduces low correlation among the individual trees by selecting a random subset of the features at each split boundary of the component trees. The Bayesian method uses a precision of 0.1. The ANN is implemented as a two-layer perceptron, with default weights = 0 and the default bias = 0.

E. Dimensionality Reduction: Significance of the Model Parameters

We apply dimensionality reduction [15] to determine the features (parameters) that are relevant to the problem, and have low correlation among themselves. We use the following techniques to detect any redundant feature that can be omitted without losing information. Rank features obtains the most significant features by computing different criterion functions. It is applied as a filtering (or a preprocessing) step before the training phase. On the other hand, sequential feature

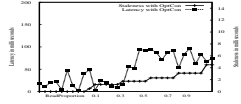
Parameter	Rank Features Technique	Sequential Attribute Selection Technique	Misclassification Error For Random Forest
C	4.81	1	0.22
RW	0.9	1	1.11
P	2.78	1	0.24

TABLE II: Significance of the Model Parameters: As per descending order of Rank Features and Sequential Attribute Selection, and ascending order of Misclassification error

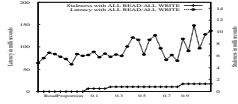
selection is a wrapper technique, that sequentially selects the features until there is no substantial improvement (given by the deviance of the fitted models) in the predictive power for an included feature. For the random forest model, we evaluate the significance of the features in the ensemble of component trees using the misclassification error. Table II presents the results of the above techniques on the model parameters C , RW , and P . Sequential feature selection outputs 1 for all the model parameters, indicating that all the parameters are significant. Rank features ranks the relative significance of the features in the order C , P , and RW . Misclassification error ranks the features in the order RW , P , and C . The order of the relative significance is different with each dimensionality reduction technique; the developer must choose the suitable ranking approach based on the learning technique. For example, the misclassification error criterion is to be used for the random forest technique.

F. Comparison of the Predictive Power of the Learning Techniques Using Model Selection Metrics

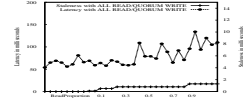
1) *The Model Selection Metrics Used:* Cross validation error (CV error) [15] is the most common and widely used model selection metric. It represents the accuracy of the model with respect to an unseen test dataset, which is different from the original dataset used to train the model. Using 10-fold cross validation, we partition the dataset into 10 subsamples, and run 10 rounds of validation. In each round a different subsample is used as the test dataset, while we train the model with the rest of the dataset. We compute the average of the mean squared error values for all validation rounds to obtain the mean CV error (Table III). We also use the Akaike Information Criterion (AIC) [7] which quantifies the quality of the generated model in terms of the information loss during the training phase. For obtaining AIC, we first compute the likelihood of each observed outcomes in the test dataset with respect to the model. We compute the maximum log likelihood L , i.e., the maximum of natural logarithms over the likelihood of each of the observed outcomes. AICC [7] (Table III) further improves upon AIC, penalizing overfitting with a correction term for finite sample size n . Thus $AICC = 2k - 2 \ln L + \frac{2k(k+1)}{n-k-1}$, where k is the number of model parameters. Apart from AICC, we also compute the Bayesian Information Criterion (BIC) [7] that uses the Bayesian prior probability estimates to penalize overfitting. Thus, $BIC = 2k \ln N - 2 \ln L$, where the additional parameter N for the sample size enables BIC to assign a greater penalty on the sample size than AICC. At smaller sample size, BIC puts lower penalty on the free parameters than AICC, whereas the penalty is higher for larger sample size (because it uses $k \ln N$ instead of k). Normally, the BIC and AICC scores (Table III) are used together to compare the models, AICC detects the problem of overfitting and BIC



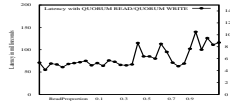
(a) Operations With Consistency Levels Predicted by OptCon satisfy the subSLA in 100% cases



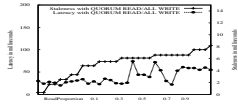
(b) Operations With READ ALL/WRITE ALL satisfy the subSLA in 70% cases



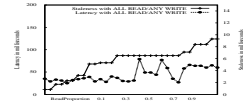
(c) Operations With READ ALL/WRITE QUORUM satisfy the subSLA in 75% cases



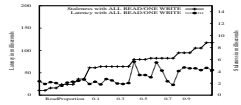
(d) Operations With READ QUORUM/WRITE QUORUM satisfy the subSLA in 75% cases



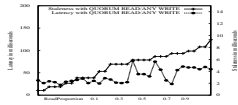
(e) Operations With READ QUORUM/WRITE ALL satisfy the subSLA in 35% cases



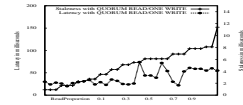
(f) Operations With READ ALL/WRITE ANY satisfy the subSLA in 30% cases



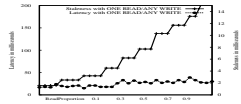
(g) Operations With READ ALL/WRITE ONE satisfy the subSLA in 30% cases



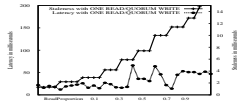
(h) Operations With READ QUORUM/WRITE ANY satisfy the subSLA in 40% cases



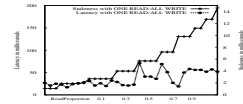
(i) Operations With READ QUORUM/WRITE ONE satisfy the subSLA in 40% cases



(j) Operations With READ ONE/WRITE ANY satisfy the subSLA in 45% cases

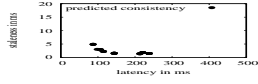


(k) Operations With READ ONE/WRITE QUORUM satisfy the subSLA in 33% cases

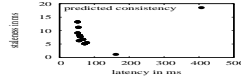


(l) Operations With READ ONE/WRITE ALL satisfy the subSLA in 48% cases

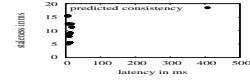
Fig. 2: Adaptability of OptCon to Varying Workload (Read Proportion): Operations done with OptCon vs operations done with manually chosen consistency levels, under the subSLA SLA-1 (Latency:250ms Staleness: 5ms)



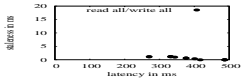
(a) Operations With Consistency Levels Predicted by OptCon under subSLA-1: Latency:250ms Staleness: 5ms



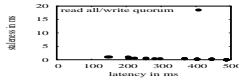
(b) Operations With Consistency Levels Predicted by OptCon under subSLA-2: Latency:100ms Staleness: 10ms



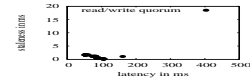
(c) Operations With Consistency Levels Predicted by OptCon under subSLA-3: Latency:20ms Staleness: 20ms



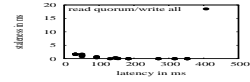
(d) Operations With READ ALL/WRITE ALL



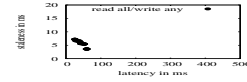
(e) Operations With READ ALL/WRITE QUORUM



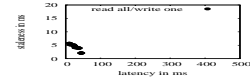
(f) Operations With READ QUORUM/WRITE QUORUM



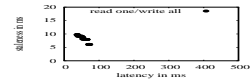
(g) Operations With READ QUORUM/WRITE ALL



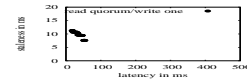
(h) Operations With READ ALL/WRITE ANY



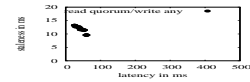
(i) Operations With READ ALL/WRITE ONE



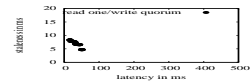
(j) Operations With READ ONE/WRITE ALL



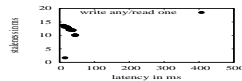
(k) Operations With READ QUORUM/WRITE ONE



(l) Operations With READ QUORUM/WRITE ANY



(m) Operations With READ ONE/WRITE QUORUM



(n) Operations With READ ONE/WRITE ANY

Consistency Level	M Value
subSLA-1	
WRITE ANY/READ ONE	68
READ/WRITE QUORUM	41
Predicted Consistency	85
subSLA-2	
READ QUORUM/WRITE ALL	70
Predicted Consistency	92
subSLA-3	
READ ALL/WRITE QUORUM	0
Predicted Consistency	100

(o) Chart Showing M-statistic values

Fig. 3: Adaptability of OptCon to Different subSLAs: Operations done with OptCon vs operations done with manually chosen consistency levels

indicates underfitting. Another criterion for the choice of the algorithm is the average prediction overhead (refer Section V-C), which is given in the last column of Table III.

Approach	Cross Validation Error	AICC	BIC	Overhead (ms)
Decision Tree	0.14	10.73	51.44	1
Bayesian Learning	0.57	12.85	53.57	1.2
Logistic Regression	1.98	16.32	57.07	0.7
Random Forest	0.14	9.51	50.24	1.3
Neural Network	0.059	12.85	63.54	1.5

TABLE III: Model Selection Results: As per descending order of AICC and BIC, and ascending order of CV and Overhead

2) *Insights: Evaluation of Learning Techniques With Respect to the Metrics:* Table III gives the CV error, the AICC score, the BIC score, and the average prediction overhead, for each of the learning algorithms that OptCon has used. The above table can guide the developers in making an informed choice regarding the correct learning technique to use. The following analysis with respect to the Table III can act as lessons learnt for future practitioners and researchers looking to apply learning techniques to solve similar problems. Our problem can be directly cast as classification of the given training data into classes labelled (Section IV-D) by the consistency levels. Hence, Decision Tree yields high accuracy and speed (Table III). We observed near random predictions with Linear regression (hence we omitted the results), because: 1) our problem is more of a classification problem, as already explained, and 2) linear regression requires the assumption of a linear model. Logistic regression fairs worst among the approaches as indicated by the values of the metrics. This is because it also treats the problem as a regression problem, whereas ours is a nonlinear classification problem. But it is still useful, since it yields a simple relation which expresses the effect of the parameters RW , Tc , and P , on L and S for an operation, under different consistency levels. Also, it can be used to determine the strength of the relationship among the parameters. Thus it can act as a basis for complex modelling algorithms. Random forest further eliminates errors due to overfitting and noisy data by: 1) bootstrapping multiple learners, and 2) using randomized feature selection. Hence, it produces the best accuracy. However, the prediction overhead due to model complexity, might increase the latency overhead beyond the thresholds of real-time subSLA. Similarly, though ANN yields high accuracy (Table III), the additional overhead due to the model complexity (it produces the most complex model) may overshoot the threshold for real-time use cases. Bayesian method requires several approximating assumptions which result in high error scores.

Decision Tree	Bayesian Learning	Logistic Regression	Random Forest	ANN
0.62	0.46	0.27	0.52	0.49

TABLE IV: Evaluation of the Accuracy-Speed Tradeoff

3) *Selection of Algorithm Based on Tradeoff Between Accuracy And Speed:* We define a measure $Perf$ for selecting the learning technique that provides an optimal tradeoff between the accuracy and speed (Table III). We assign the overhead of the slowest learning technique (as per Table III) as the baseline overhead O_{base} . Since ANN has the maximum overhead (1.5 ms) as per Table III, $O_{base} = 1.5$. The speedup $Speedup$

obtained with a chosen learning algorithm relative to the slowest algorithm is estimated from the observed relative decrease in overhead for the chosen algorithm with respect to the baseline O_{base} (i.e., for the slowest algorithm, $Speedup = 0$). Thus, $Speedup = \frac{O_{base} - O}{O_{base}}$, where O is the overhead of the chosen technique. We denote the prediction error for a given learning technique as E . We assign the error for the least accurate algorithm, as per Table III, as the baseline E_{base} . Logistic regression has the maximum CV error (1.98 as per Table III). Hence, $E_{base} = 1.98$. The value of E and E_{base} depends on the choice of the error measure. The developer can either use the CV error, or both AICC and BIC taken together. In the latter case, E is given as the maximum between the AICC and BIC for the chosen algorithm, and E_{base} is given as the maximum between the AICC and BIC for the least accurate algorithm. We compute the relative increase in accuracy (A_{Rel}) obtained with a chosen learning algorithm with respect to the least accurate algorithm. A_{Rel} is estimated from the proportion of observed decrease in error for the chosen algorithm with respect to the baseline E_{base} . Thus, $A_{Rel} = \frac{E_{base} - E}{E_{base}}$. Finally, the measure $Perf$ is computed as $Perf = w_{Speedup} \times Speedup + w_A \times A_{Rel}$, where $w_{Speedup}$ and w_A are the weights (real numbers in the closed interval $[0, 1]$, such that $w_{Speedup} + w_A = 1$) assigned by the developers to quantify the relative importance of speed and accuracy, respectively, for selecting a learning technique. The developer can assign a larger weight to accuracy (i.e., $w_A > w_{Speedup}$), if accuracy is more important than speedup (and vice versa) according to the use case. The developer chooses the learning algorithm that produces the maximum value of $Perf$, with a given choice of error measure. We provide the values of $Perf$ for various learning algorithms in Table IV, with CV Error as the error measure, and equal weights assigned to accuracy and speedup (i.e., $w_A = w_{Speedup} = 0.5$). The value of $Perf$ is maximum for the decision tree algorithm, implying that it produces an optimal tradeoff of accuracy and speed according to our choice of error measure, and the assigned values for the weights w_A and $w_{Speedup}$. In most cases, the difference in overhead between algorithms is negligibly small, hence the $Speedup$ is not a criterion of significance for most cases. Thus, generally developers would assign weights $w_A = 1$ and $w_{Speedup} = 0$, choosing the algorithm with the highest accuracy.

G. Adaptability of OptCon to Varying Workload

We demonstrate the adaptability of OptCon to a varying workload, by tuning the read proportion (i.e., proportion of reads) parameter (RW) in the YCSB benchmark workloads (Figure 2(a)). As per Table IV, we choose the Decision tree implementation of the Learner module. We compare operations performed with OptCon, with those performed with manually chosen consistency levels. In the Figures 2(a) through 2(l), read proportion is plotted along the x axis, the observed latency along the primary y axis, and the staleness along the secondary y axis. We demonstrate that, for a specific read proportion, only a subset of all possible read-write consistency levels is *optimal*, i.e., satisfies latency and staleness thresholds in a given subSLA. Varying the read proportion (RW) in the workload from 0.1 to 1, we observe that a manually chosen read-write consistency level is optimal for only a fraction of the total number of cases, under the subSLA SLA-

1 (Table I). OptCon demonstrates its adaptability to varying read proportion, applying the optimal consistency level for each read proportion, thus satisfying the subSLA in 100% cases. OptCon can adapt to changing read proportion due to the presence of the parameter RW as a feature in the prediction model \mathcal{M} . As shown by the results in [16], the frequency of stale results (i.e., higher Γ scores) increases with increasing read proportion in the workload. Hence with higher read proportion in the right-half of the x axis, stronger read consistency levels are required to return less stale results. Also, since the proportion of writes is less, the penalty for the write latency overhead is less. Hence stronger write consistency levels, that result in high write latency, are acceptable. Thus for higher read proportions, combinations of strong read-write consistency levels, i.e., ALL/QUORUM, succeed in lowering staleness values to acceptable bounds (right-half of the Figures 2(b), 2(c), 2(d), and 2(e)). Strong consistency levels achieve a maximum success rate of 75% in satisfying the subSLA, with READ ALL/QUORUM WRITE (Figure 2(c)) and READ QUORUM/WRITE ALL (Figure 2(d)). Taking into account the clock skew, the staleness is effectively 0 in these cases. For the same reasons, for higher read proportions, weak read consistency levels result in failure to achieve stringent staleness thresholds, as demonstrated by the points in the right-half of Figures 2(f) through 2(l). With lower read proportions in the left-half of the x axes, the frequency of stale read results are smaller [16]. In this case, weaker manually chosen read-write consistency levels, i.e., ONE/ANY (left-half of Figures 2(f) through 2(l)) are sufficient for returning consistent results. For lower read proportions, stronger consistency levels, i.e., ALL or QUORUM, are unnecessary, only resulting in high latency overhead. With weak consistency levels, we observe a maximum success rate of only $\leq 55\%$ in satisfying the subSLA (left-half of Figures 2(b) through 2(e)), with READ ONE/WRITE ALL (Figure 2(l)). Thus, manually chosen consistency levels show a maximum success rate of 75% in satisfying the subSLA, in contrast with a 100% success rate demonstrated by OptCon (Figure 2(a)). Thus, OptCon is more effective than any manually chosen consistency level in adapting to workload variations.

H. Adaptability of OptCon to Different subSLAs

For a given subSLA, only a subset of all possible manually chosen consistency levels is *optimal*, i.e., satisfies the subSLA thresholds. Consistency levels predicted by OptCon are always matching, i.e., it always chooses from the above optimal set of consistency levels for the given subSLA. We have integrated OptCon (using Decision tree learning as per Table IV) with the RUBBoS [29] benchmark, that can simulate concurrent access, and interleaving user access. Figures 3(a) through 3(n) plot the observed staleness along the y axis, and latency along the x axis, under subSLAs SLA-1, SLA-2 and SLA-3, respectively (Table I). Figures 3(a) through 3(c) demonstrate experiments performed with OptCon. Figures 3(d) through 3(j) correspond to experiments done with manually chosen consistency levels. We demonstrate a few extreme cases of occasional heavy network traffic in real world applications with artificially introduced network delays (refer to the few boundary cases with arbitrarily high latency in Figures 3(a), 3(b), and 3(c), that violate the respective subSLAs). These boundary cases were simulated using the traffic shaping feature of Traffic

Control [18], a linux-based tool for configuring network traffic, that controls the transmission rate by lowering the network bandwidth. SLA-1 (Table I) represents systems demanding stronger consistency. Manually chosen weak read-write consistency level, i.e., ANY/ONE, fails to satisfy the stringent staleness bound of SLA-1 (Figures 3(k) through 3(m)). On the other hand, strong consistency levels (i.e., ALL) (Figures 3(d) through 3(g)) satisfy SLA-1. OptCon satisfies SLA-1 by applying the respective optimal consistency levels (i.e., ALL in this case) for SLA-1 (Figure 3(a)). For lower latency bound in SLA-3 (Table I), the manually chosen strong consistency levels (ALL/QUORUM) unnecessarily produce latencies beyond the acceptable threshold (Figures 3(d) through 3(g)). However, weaker consistency levels (i.e., ONE/ANY) prove to be optimal (Figures 3(k) through 3(j)). Again, OptCon succeeds by choosing the respective optimal consistency levels (i.e., ONE/ANY in this case) for SLA-3 (Figure 3(c)). Similarly with SLA-2, a subset of manually chosen fixed consistency levels (Figures 3(h), 3(i), and 3(j)) produce optimal results, whereas OptCon successfully achieves SLA-2 in Figure 3(b) for all operations. We evaluate OptCon by a measure M , which measures the adaptability of the system with the subSLA. Following [35], M (Table 3(o)) computes the percentage of cases which did not violate the subSLA. For all the given subSLAs, the M values for operations performed with OptCon, given in the Figures 3(a), 3(b), and 3(c), exceed the M for operations using fixed consistency levels. Thus, OptCon is at least as effective as any optimal consistency level for the given subSLA. Moreover, OptCon produces matching choices where fixed consistency levels fail.

VI. RELATED WORK

Wada et al. [37] and Bermbach et al. [3] analyze and quantify consistency from a system-centric perspective that focuses on the convergence time of the replication protocol. Bailis et al. [2] and Rahman et al. [30] instead consider a client-centric perspective of consistency, in which a consistency anomaly occurs only if differences in state among replicas lead to a client application actually observing a stale read. In practice eventual consistency is preferred over strong consistency in scenarios where the system must maintain availability during network partitions [4, 36], or when the application is latency-sensitive and able to tolerate occasional inconsistencies [6]. The state machine replication paradigm achieves the strongest possible form of consistency by physically serializing operations [23]. Lamport's Paxos protocol is a quorum-based fault-tolerant protocol for state machine replication [25]. Mencius improves upon Paxos by ensuring better scalability and higher throughput under high client load using a rotating coordinator scheme [27]. Using variations of Paxos, a number of systems [31, 5, 20, 12] claim to provide scalability and fault-tolerance. Relatively few systems [39, 35, 1, 33] provide mechanisms for fine-grained control over consistency. Li et al. [26] applies static analysis for automated consistency tuning for relational databases.

VII. CONCLUSIONS

OptCon automates client-centric consistency-latency tuning in quorum-replicated stores, based on staleness and latency thresholds in the SLAs. OptCon can adapt to variations in the

workload, and is at least as effective as any manually chosen consistency setting in satisfying a given SLA. OptCon provides insights for applying machine learning to similar problems.

REFERENCES

- [1] M. S. Ardekani and D. B. Terry. A self-configurable geo-replicated cloud storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 367–381, Broomfield, CO, Oct. 2014. USENIX Association.
- [2] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB Endow.*, 5(8):776–787, Apr. 2012.
- [3] D. Bernbach and S. Tai. Eventual consistency: How soon is eventual? An evaluation of Amazon S3’s consistency behavior. In *Proc. Workshop on Middleware for Service Oriented Computing (MW4SOC)*, 2011.
- [4] K. Birman and R. Friedman. *Trading Consistency for Availability in Distributed Systems*. Cornell University. Department of Computer Science, 1996.
- [5] W. J. Bolosky, D. Bradshaw, R. B. Haagens, N. P. Kusters, and P. Li. Paxos replicated state machines as the basis of a high-performance data store. In *Proc. of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI’11*, pages 11–11, Berkeley, CA, USA, 2011. USENIX Association.
- [6] E. A. Brewer. Towards robust distributed systems (Invited Talk). In *Proc. of the 19th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2000.
- [7] K. P. Burnham and D. R. Anderson. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer Science & Business Media, 2002.
- [8] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP ’11*, pages 143–157, New York, NY, USA, 2011. ACM.
- [9] P. Cassandra. Apache cassandra use cases. <http://planetcassandra.org/apache-cassandra-use-cases/>, 2015.
- [10] A. Cockcroft and D. Sheahan. Benchmarking cassandra scalability on AWS - over a million writes per second. <http://techblog.netflix.com/2011/11/>, 2011.
- [11] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC ’10*, pages 143–154, New York, NY, USA, 2010. ACM.
- [12] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google’s globally-distributed database. In *Proc. of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI’12*, pages 251–264, Berkeley, CA, USA, 2012. USENIX Association.
- [13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Voshall, and W. Vogels. Dynamo: Amazon’s highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, Oct. 2007.
- [14] T. Everts. Web Performance Today. <http://www.webperformancetoday.com/2012/04/02/>, 2012.
- [15] P. Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, New York, NY, USA, 2012.
- [16] W. M. Golab, M. R. Rahman, A. AuYoung, K. Keeton, J. J. Wylie, and I. Gupta. Client-centric benchmarking of eventual consistency for cloud storage systems. In *ICDCS*, page 28, 2014.
- [17] C. Hale and R. Kennedy. Using Riak at Yammer. http://dl.dropbox.com/u/2744222/2011-03-22_Riak-At-Yammer.pdf, 2011.
- [18] B. Hubert, T. Graf, G. Maxwell, R. Van Mook, M. Van Oosterhout, P. B. Schroeder, J. Spaans, and P. Larroy. *Linux Advanced Routing & Traffic Control HOWTO*. Linux Advanced Routing & Traffic Control, <http://lartc.org/>, Apr. 2004.
- [19] A. Jain. Using cassandra for real-time analytics: Part 1. <http://blog.markedup.com/2013/03/cassandra-real-time-analytics/>, 2011.
- [20] T. Kraska, G. Pang, M. J. Franklin, S. Madden, and A. Fekete. MDCC: multi-data center consistency. In *Proc. of the 8th ACM European Conference on Computer Systems, EuroSys ’13*, pages 113–126, New York, NY, USA, 2013. ACM.
- [21] K. LaCurtis, J. C. Mogul, H. Balakrishnan, and Y. Turner. Cicada: Introducing Predictive Guarantees for Cloud Networks. In *HotCloud*, June 2014.
- [22] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.
- [23] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558, 1978.
- [24] L. Lamport. On interprocess communication. *Distributed Computing*, 1(2):77–85, 1986.
- [25] L. Lamport. Paxos made simple, fast, and byzantine. In *OPODIS*, pages 7–9, 2002.
- [26] C. Li, J. Leitão, A. Clement, N. Preguiça, R. Rodrigues, and V. Vafeiadis. Automating the choice of consistency levels in replicated systems. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 281–292, Philadelphia, PA, June 2014. USENIX Association.
- [27] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: Building efficient replicated state machines for WANs. In *Proc. of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI’08*, pages 369–384, Berkeley, CA, USA, 2008. USENIX Association.
- [28] C. Meiklejohn. Riak PG: Distributed process groups on dynamo-style distributed storage. In *Proc. of the Twelfth ACM SIGPLAN Workshop on Erlang, Erlang ’13*, pages 27–32, New York, NY, USA, 2013. ACM.
- [29] Objectweb Consortium. RUBBoS: Bulletin Board Benchmark. <http://jmob.ow2.org/rubbos.html>, 2007.
- [30] M. R. Rahman, W. , A. AuYoung, K. Keeton, and J. J. Wylie. Toward a principled framework for benchmarking consistency. In *Proc. of the Eighth USENIX Conference on Hot Topics in System Dependability, HotDep’12*, pages 8–8, Berkeley, CA, USA, 2012. USENIX Association.
- [31] J. Rao, E. J. Shekita, and S. Tata. Using paxos to build a scalable, consistent, and highly available datastore. *PVLDB*, 4(4):243, 2011.
- [32] L. Ravindranath, J. Padhye, R. Mahajan, and H. Balakrishnan. Timercard: Controlling user-perceived delays in server-based mobile applications. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP ’13*, pages 85–100, New York, NY, USA, 2013. ACM.
- [33] K. Sivaramakrishnan, G. Kaki, and S. Jagannathan. Declarative programming over eventually consistent data stores. *SIGPLAN Not.*, 50(6):413–424, June 2015.
- [34] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah. Serving large-scale batch computed data with project Voldemort. In *Proc. of the 10th USENIX Conference on File and Storage Technologies (FAST)*, 2012.
- [35] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *Proc. of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP ’13*, pages 309–324, New York, NY, USA, 2013. ACM.
- [36] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, pages 172–182, 1995.
- [37] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu. Data consistency properties and the trade-offs in commercial cloud storage: the consumers’ perspective. In *Proc. Conference on Innovative Data Systems Research (CIDR)*, pages 134–143, 2011.
- [38] K. Winstein and H. Balakrishnan. Tcp ex machina: Computer-generated congestion control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM ’13*, pages 123–134, New York, NY, USA, 2013. ACM.
- [39] H. Yu and A. Vahdat. Building replicated internet services using TACT: A toolkit for tunable availability and consistency tradeoffs. In *WECWIS*, pages 75–84, 2000.